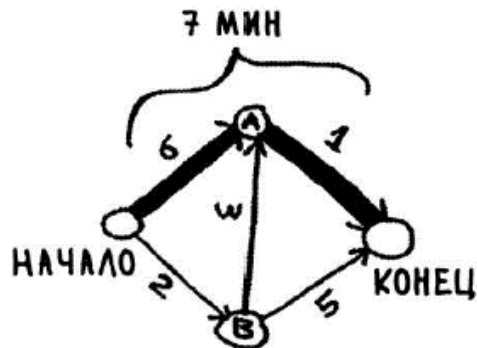
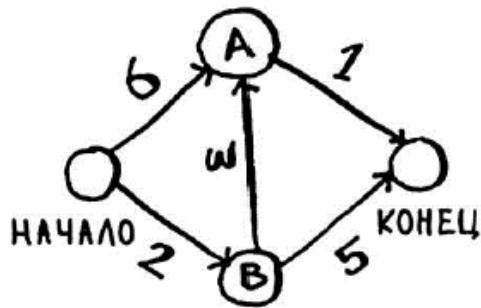
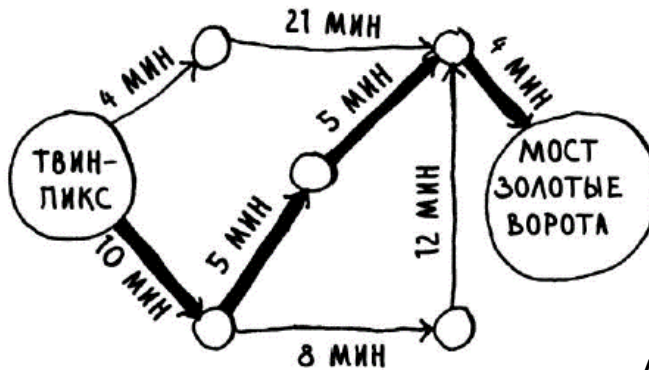


Алгоритм Дейкстри



Алгоритм Дейкстри складається з чотирьох кроків:

1. Знайти вузол з найменшою вартістю (тобто вузол, до якого можна дістатися за мінімальний час).
2. Відновити вартості сусідів цього вузла.
3. Повторювати, поки це не буде зроблено для всіх вузлів графа.
4. Обчислити підсумковий шлях.

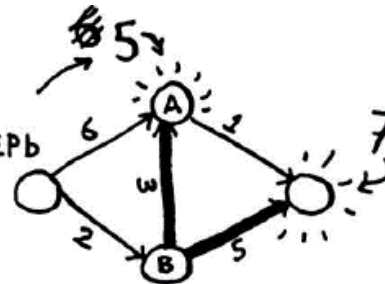
Крок 1: знайти вузол з найменшою вартістю. Ви стоїте на самому початку і думаєте, куди податися: до вузла А чи до вузла В. Скільки часу знадобиться, щоб дістатися до кожного з цих вузлів? Вузол В - найближчий (він знаходиться всього в 2 хвилини).

УЗЕЛ	ВРЕМЯ ПЕРЕХОДА К УЗЛУ
А	6
В	2
КОНЕЦ	∞

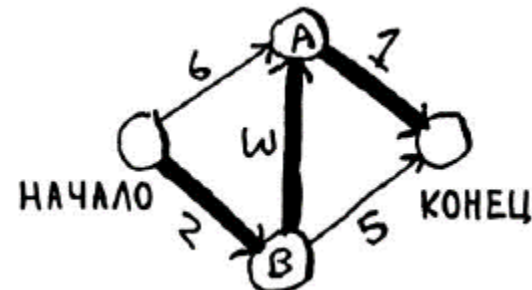
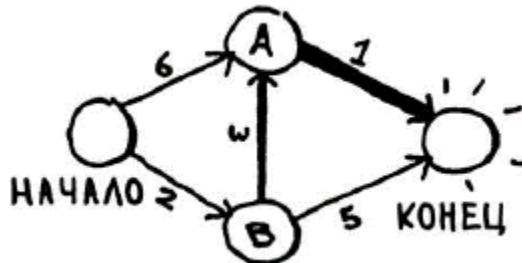
Крок 2: обчислити, скільки часу буде потрібно для того, щоб дістатися до всіх сусідів В при переході по ребру з В.

УЗЕЛ	ВРЕМЯ
А	5
В	2
КОНЕЦ	7

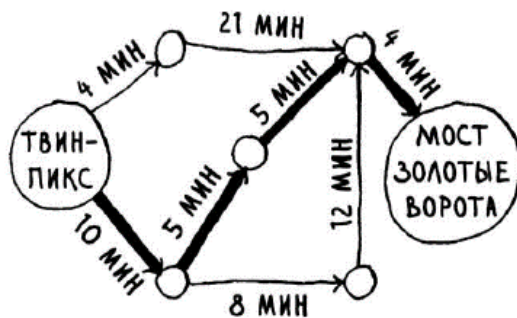
ЧТОБЫ ДО-
БРАТЬСЯ ДО
УЗЛА А, ТЕПЕРЬ
ТРЕБУЕТСЯ
ВСЕГО 5 МИН



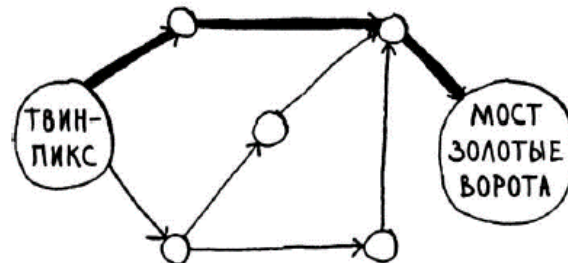
Крок 3: повторюємо! Знову крок 1: знаходимо вузол, для переходу до якого потрібно найменший час. З вузлом В робота закінчена, тому найменшу оцінку часу має вузол А. Знову крок 2: оновлюємо вартості сусідів А.



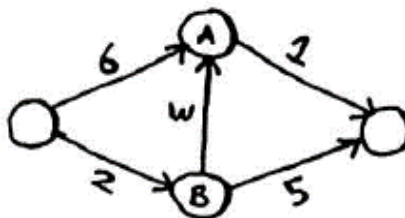
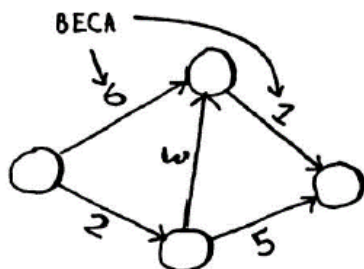
Графи



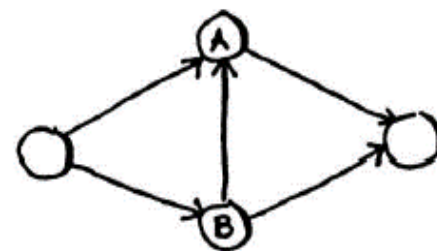
ВЗВЕШЕННЫЙ ГРАФ



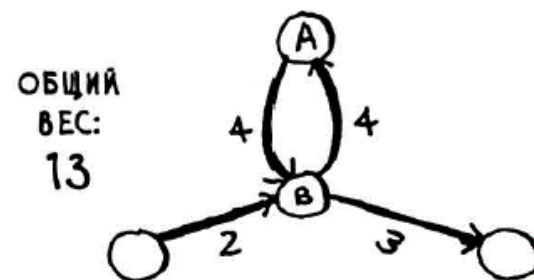
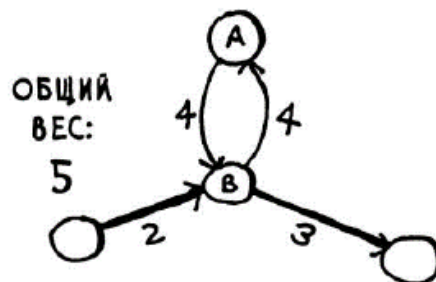
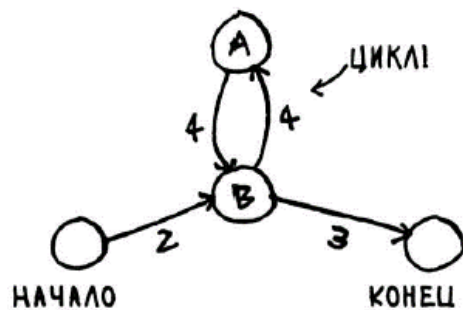
НЕВЗВЕШЕННЫЙ ГРАФ
(ПОИСК В ШИРИНУ)



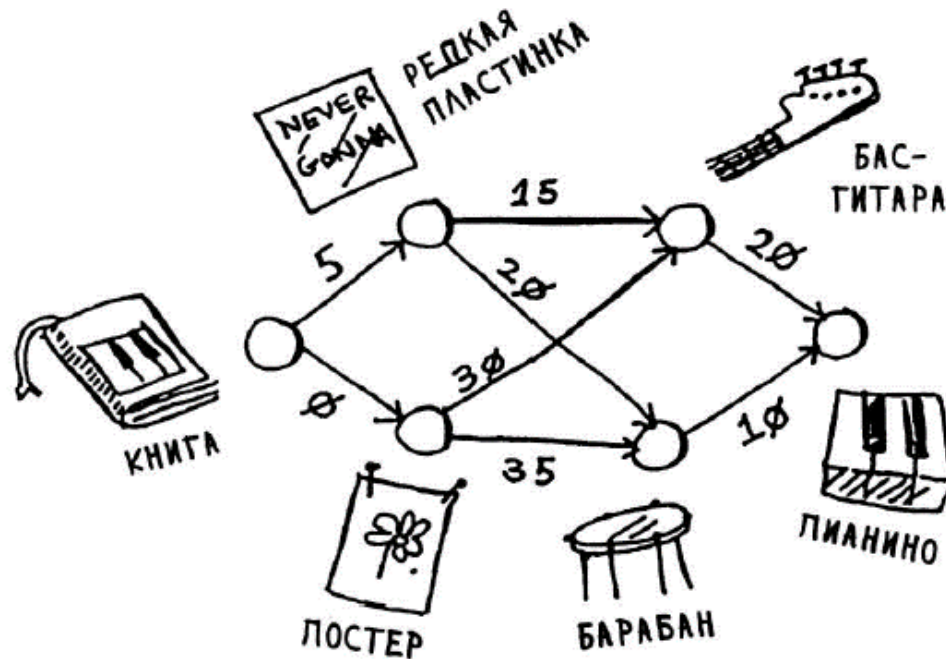
ВЗВЕШЕННЫЙ ГРАФ



НЕВЗВЕШЕННЫЙ ГРАФ



Рама хоче виміняти свою книгу по музиці на піаніно. «Я тобі дам за книгу ось цей постер, - каже Алекс. - Це моя улюблена група Destroyer. Або можу дати за книгу рідкісну платівку Ріка Естлі і ще \$ 5 ». - «О, я чула, що на цій платівці є відмінні пісні, - каже Емі. - Готова віддати за постер або пластинку мою гітару або ударну установку ». «Все життя мріяв грати на гітарі, - вигукує Бетховен. - Слухай, я віддам тобі своє піаніно за будь-яку з речей Емі». Рама з невеликими додатковими витратами може поміняти свою книгу на даний піаніно. Тепер залишається зрозуміти, як йому витратити найменшу суму на ланцюжку обмінів.



Необхідно побудувати таблицю з цінами всіх вузлів. Таблиця буде оновлюватися в міру роботи алгоритму. Для обчислення підсумкового шляху в таблицю також необхідно додати стовпець «батьки».

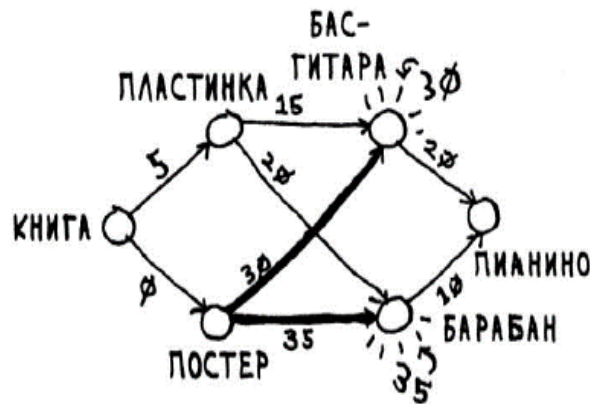
УЗЕЛ	СТОИМОСТЬ
ПЛАСТИНКА	5
ПОСТЕР	0
ГИТАРА	∞
БАРАБАН	∞
ПИАНИНО	∞

МЫ ЕЩЕ НЕ ДОХОДИЛИ ДО ЭТИХ УЗЛОВ ОТ НАЧАЛЬНОГО

УЗЕЛ	РОДИТЕЛЬ
ПЛАСТИНКА	КНИГА
ПОСТЕР	КНИГА
ГИТАРА	—
БАРАБАН	—
ПИАНИНО	—

Крок 1: знайти вузол з найменшою вартістю. В даному випадку найдешевший варіант обміну з доплатою \$ 0 - це постер.

Крок 2: Обчислити, скільки часу буде потрібно для того, щоб дістатися до всіх його сусідів (вартість).

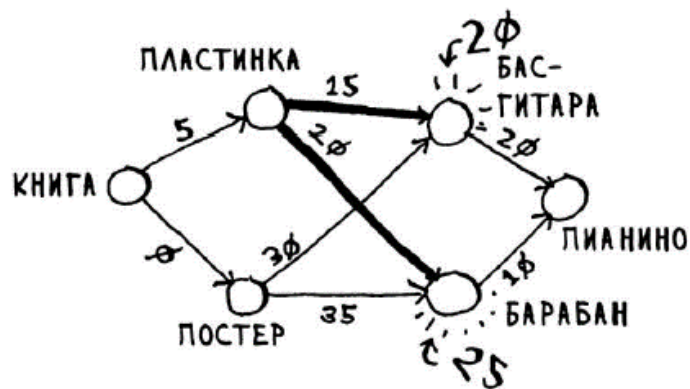


РОДИТЕЛЬ	УЗЕЛ	СТОИМОСТЬ
КНИГА		5
КНИГА	ПОСТЕР	0
ПОСТЕР		0 30
ПОСТЕР	БАРАБАН	0 35
—	ПИАНИНО	∞

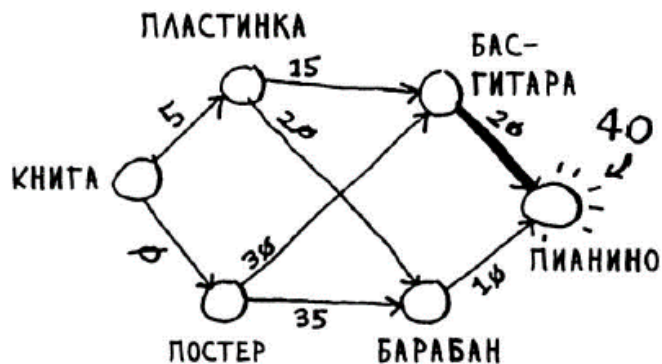
К ЭТИМ УЗЛАМ
ПЕРЕХОДИМ ОТ
УЗЛА «ПОСТЕР»

РОДИТЕЛЬ	УЗЕЛ	СТОИ- МОСТЬ
КНИГА	ПЛАСТИНКА	5
КНИГА	ПОСТЕР	∅
ПОСТЕР	ГИТАРА	3∅
ПОСТЕР	БАРАБАН	35
—	ПИАНИНО	∞

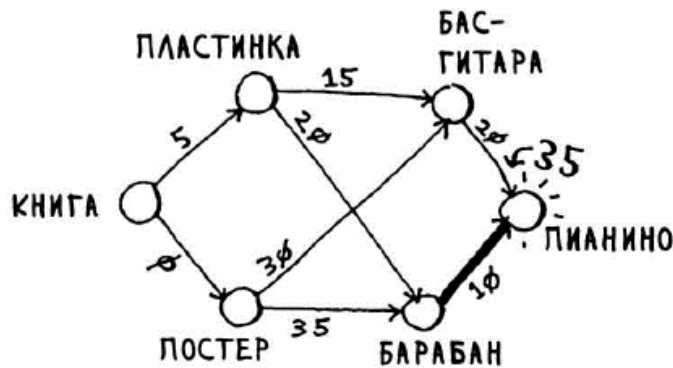
Знову крок 1: платівка - наступний за вартістю вузол (\$ 5).
Знову крок 2: оновлюються значення всіх його сусідів.



РОДИТЕЛЬ	УЗЕЛ	СТОИ- МОСТЬ
КНИГА	ПЛАСТИНКА	5
КНИГА	ПОСТЕР	∅
ПЛАСТИНКА	ГИТАРА	∅ 2∅
ПЛАСТИНКА	БАРАБАН	∅ 25
—	ПИАНИНО	∞



РОДИТЕЛЬ	УЗЕЛ	СТОИ- МОСТЬ
КНИГА		5
КНИГА		∅
ПЛАСТИНКА		2∅
ПЛАСТИНКА		25
ГИТАРА		4∅

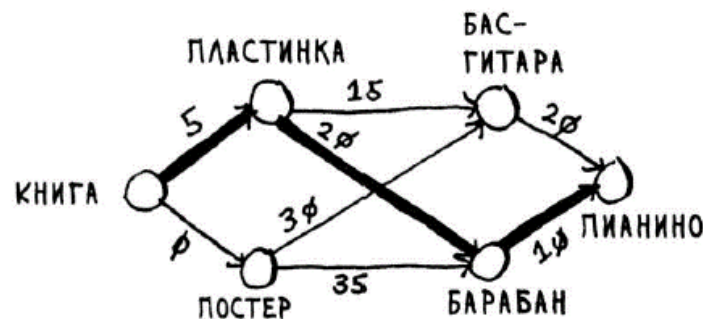
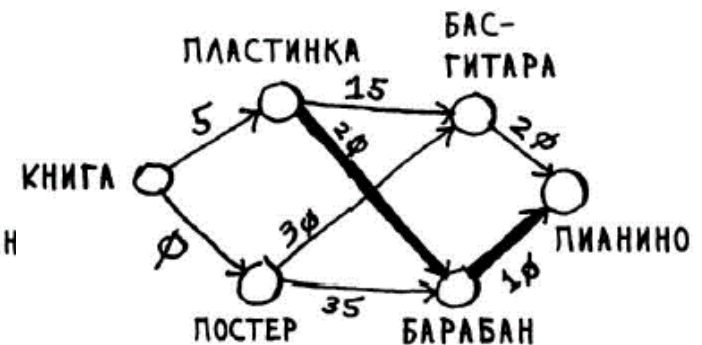
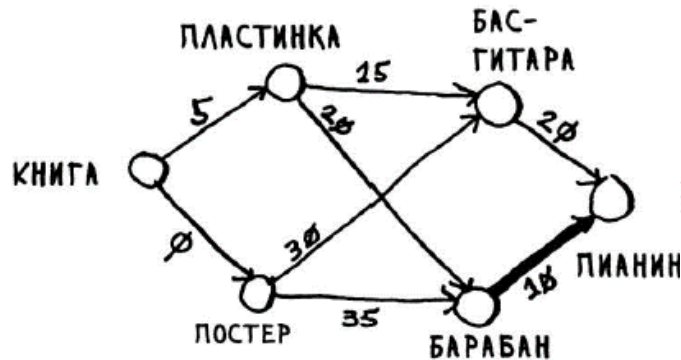


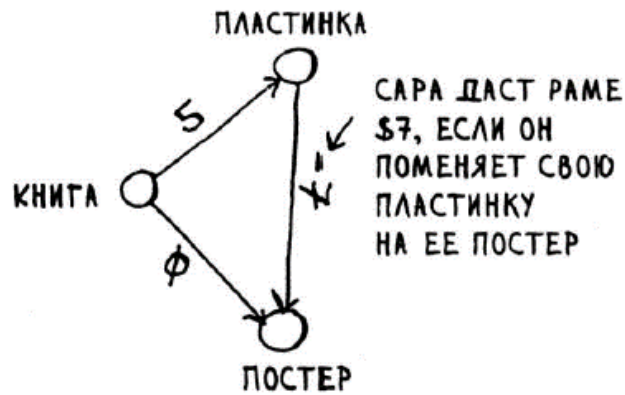
РОДИТЕЛЬ	УЗЕЛ	СТОИ-МОСТЬ
КНИГА	ПЛАСТИНКА	5
КНИГА	ПОСТЕР	0
ПЛАСТИНКА	ГИТАРА	20
ПЛАСТИНКА	БАРАБАН	25
БАРАБАН	ПИАНИНО	<u>35</u>

Рама може отримати піаніно ще дешевше, помінявши ударну установку на піаніно. Таким чином, найдешевша ланцюжок обмінів обійдеться Рамі в \$ 35, але як цей шлях визначити? Для початку візьмемо батька вузла «піаніно».

→

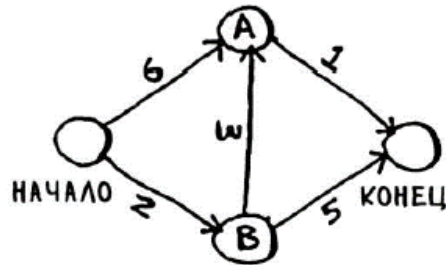
РОДИТЕЛЬ	УЗЕЛ
КНИГА	ПЛАСТИНКА
КНИГА	ПОСТЕР
ПЛАСТИНКА	ГИТАРА
ПЛАСТИНКА	БАРАБАН
БАРАБАН	ПИАНИНО





Якщо застосувати алгоритм Дейкстри до цього графу, Рама вибере невірний шлях. Він піде по більш довгому шляху. Алгоритм Дейкстри не може використовуватися при наявності ребер, що мають негативну вагу. Такі ребра порушують роботу алгоритму. Якщо ви хочете знайти найкоротший шлях в графі, що містить ребра з негативним вагою, для цього існує спеціальний алгоритм, званий алгоритмом **Белмана - Форда**.

Реалізація



НАЧАЛО	A	6
	B	2
A	КОНЕЦ	7
B		5
КОНЕЦ		—

ГРАФ
(GRAPH)

Для реалізації цього прикладу знадобляться три хеш - таблиці. Хеш-таблиці вартостей і батьків будуть оновлюватися по ходу роботи алгоритму

A	6
B	2
КОНЕЦ	∞

СТОЙМОСТИ
(COSTS)

A	НАЧАЛО
B	НАЧАЛО
КОНЕЦ	—

РОДИТЕЛИ
(PARENTS)

Спочатку необхідно реалізувати граф. Для цього буде використана хеш-таблиця:

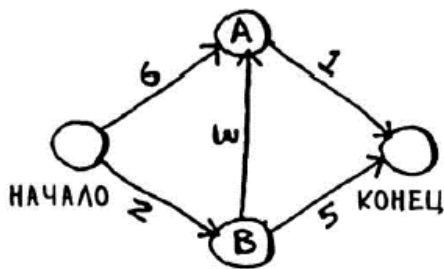
```
graph["start"] = {}  
graph["start"]["a"] = 6  
graph["start"]["b"] = 2
```

Для отримання всіх сусідів початкового вузла можна скористатися наступним виразом

```
>>> print graph["start"].keys()  
["a", "b"]
```

Якщо ви захочете дізнатися ваги цих ребер?

```
>>> print graph["start"]["a"]  
2
```



```

graph["a"] = {}
graph["a"]["fin"] = 1
graph["b"] = {}
graph["b"]["a"] = 3
graph["b"]["fin"] = 5
graph["fin"] = {} // У кінцевого вузла немає сусідів

```

Чи можна уявити нескінченність в Python? Виявляється, можна:

```
infinity = float("inf")
```

Код створення таблиці вартостей costs:

```

infinity = float("inf")
costs = {}
costs["a"] = 6
costs["b"] = 2
costs["fin"] = infinity

```

Код створення хеш-таблиці батьків:

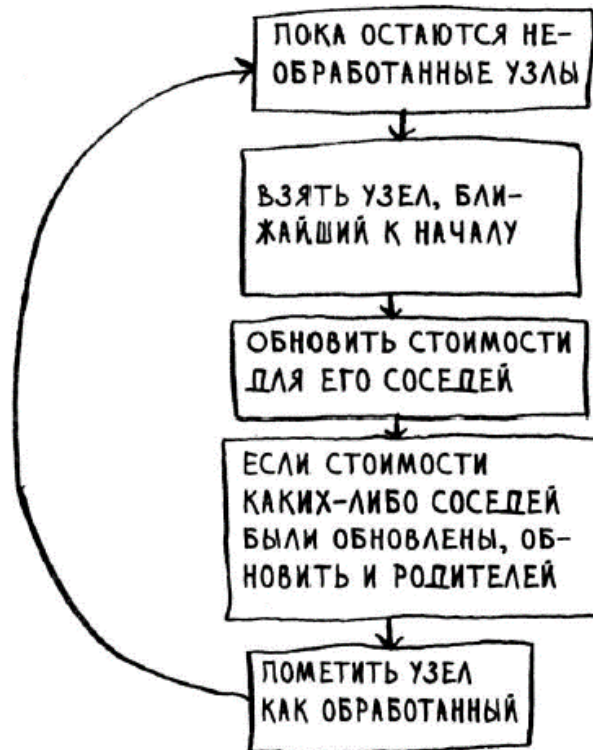
```

parents = {}
parents["a"] = "start"
parents["b"] = "start"
parents["fin"] = None

```

Нарешті, потрібен масив для відстеження всіх вже оброблених вузлів, так як один вузол не повинен оброблятися багаторазово:

```
processed = []
```



<code>node = find_lowest_cost_node(costs)</code>	←	Найти узел с наименьшей стоимостью среди необработанных
<code>while node is not None:</code>	←	Если обработаны все узлы, цикл <code>while</code> завершен
<code>cost = costs[node]</code>		
<code>neighbors = graph[node]</code>		
for <code>n in neighbors.keys():</code>	←	Перебрать всех соседей текущего узла
<code>new_cost = cost + neighbors[n]</code>		Если к соседу можно быстрее
if <code>costs[n] > new_cost:</code>	←	добраться через текущий узел...
<code>costs[n] = new_cost</code>	←	...обновить стоимость для этого узла
<code>parents[n] = node</code>	←	Этот узел становится новым родителем для соседа
<code>processed.append(node)</code>	←	Узел помечается как обработанный
<code>node = find_lowest_cost_node(costs)</code>	←	Найти следующий узел для обработки и повторить цикл

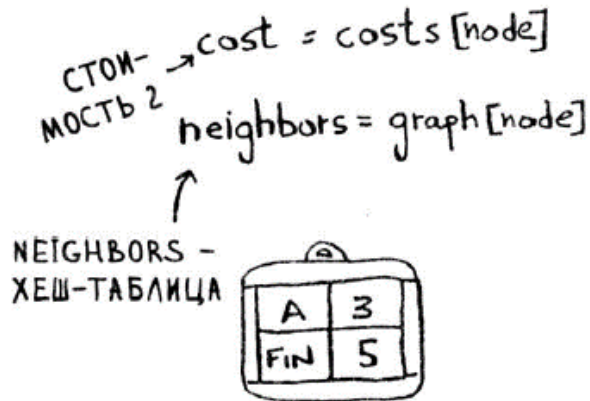
Знайти вузол з найменшою вартістю.

`node = find_lowest_cost_node(costs)`

A	6
B	2
КОНЕЦ	∞

СТОИМОСТИ

Отримати вартість і сусідів цього вузла.



START	A	6
	B	2
A	FIN	1
B	A	3
	FIN	5
FIN	—	

ГРАФ

Перебрати сусідів.



У кожного вузла є вартість, яка визначає, скільки часу буде потрібно для досягнення цього вузла від початку. Тут ми обчислюємо, скільки часу буде потрібно для досягнення вузла А по шляху **Початок > Вузол В > Вузол А**

$$\text{new_cost} = \text{cost} + \text{neighbors}[n]$$

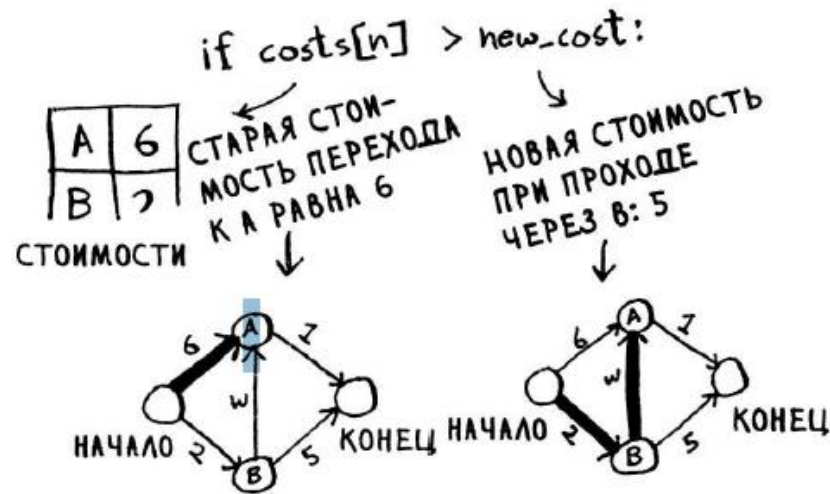
\uparrow
 СТОИМОСТЬ «В»,
 т. е. 2

\downarrow
 РАССТОЯНИЕ
 ОТ В ДО А: 3

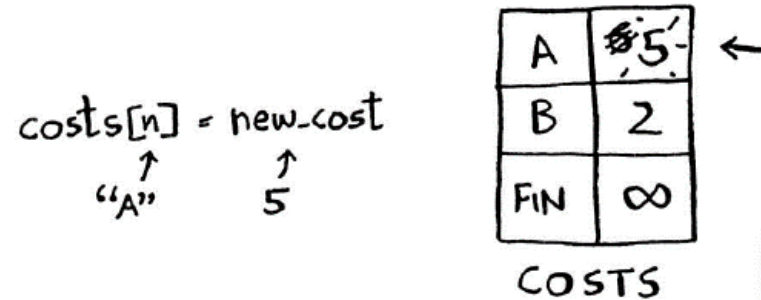
}

$\text{new_cost} = 2 + 3$
 $= 5$

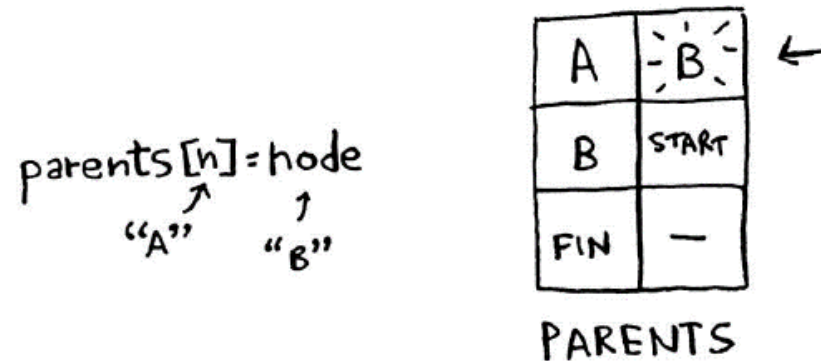
Порівняємо ці вартості:



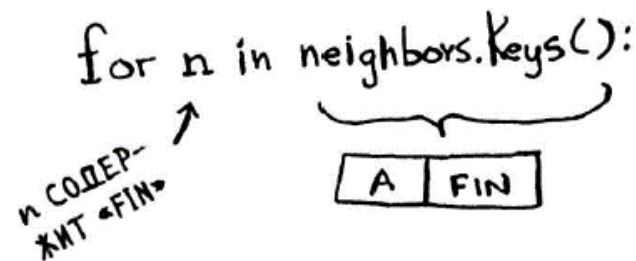
Ми знайшли більш короткий шлях до вузла A! Оновимо вартість.



Новий шлях проходить через вузол B, тому B призначається новим батьком.



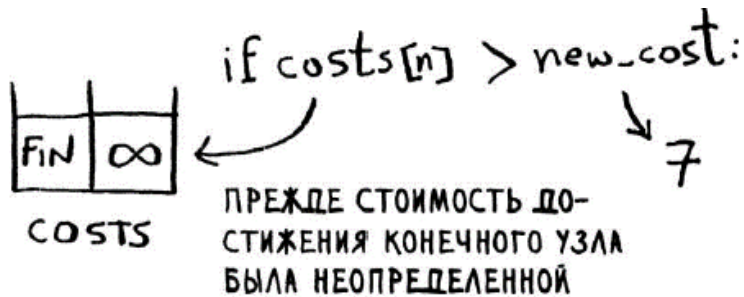
Ми знову повернулися до початку циклу. Наступним сусідом в циклі for є кінцевий вузол.



Скільки часу буде потрібно для досягнення кінцевого вузла, якщо йти через вузол В?

$$\text{new_cost} = \underset{\substack{\downarrow \\ 2}}{\text{cost}} + \underset{\substack{\downarrow \\ \text{РАССТОЯНИЕ} \\ \text{ОТ В ДО КОНЦА:} \\ 5}}{\text{neighbors}[n]} \quad \left. \vphantom{\text{new_cost}} \right\} \begin{array}{l} 2 + 5 \\ = 7 \end{array}$$

Буде потрібно 7 хвилин. Попередня вартість була нескінченною, а 7 хвилин менше нескінченності.



$$\underset{\substack{\uparrow \\ \text{"FIN"}}}{\text{costs}[n]} = \underset{\substack{\uparrow \\ 7}}{\text{new_cost}}$$

A	5
B	2
FIN	7

COSTS ←

Буде потрібно 7 хвилин. Попередня вартість була нескінченною, а 7 хвилин менше нескінченності....

$$\underset{\substack{\uparrow \\ \text{"FIN"}}}{\text{parents}[n]} = \underset{\substack{\uparrow \\ \text{"B"}}}{\text{node}}$$

A	B
B	START
FIN	B

PARENTS ←

Ми оновили вартості всіх сусідів вузла B. Вузол позначається як оброблений.

`processed.append(node)` ОБРАБОТАННЫЕ
"B" ↑ УЗЛЫ:

B

Знайти наступний вузол для обробки.

`node = find_lowest_cost_node(costs)` НЕОБРАБОТАННЫЙ
"A" ↑ УЗЕЛ С МИНИМАЛЬНОЙ СТОИМОСТЬЮ
УЖЕ ОБРАБОТАН →

A	5
B	2
FIN	7

COSTS

Отримати вартість і сусідів вузла A.

`cost = costs[node]`
5 ↑
`neighbors = graph[node]`
↑

FIN	1
-----	---

У вузла А всього один сусід: кінцевий вузол.

```
for n in neighbors.keys():  
    "FIN"
```

↑
"FIN"

└───┬───┘
FIN

Час досягнення кінцевого вузла становить 7 хвилин. Скільки часу буде потрібно для досягнення кінцевого вузла, якщо йти через вузол А?

$$\text{new_cost} = \text{cost} + \text{neighbors}[n]$$

↓ ↓

СТОИМОСТЬ
ПЕРЕХОДА К А
ОТ НАЧАЛА: 5 СТОИМОСТЬ ОТ А
ДО КОНЕЧНОГО
УЗЛА: 1

} 5 + 1
= 6

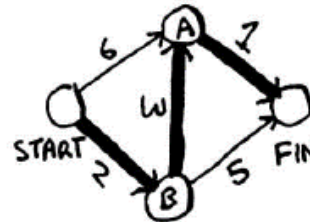
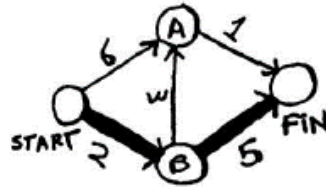
if costs[n] > new_cost:

↓ ↓

СТАРАЯ СТОИ-
МОСТЬ ПЕРЕХОДА
К КОНЕЧНОМУ
УЗЛУ: 7 СТОИМОСТЬ
ПРИ ПРОХОДЕ
ЧЕРЕЗ А: 6

D	2
FIN	7

COSTS



Через вузол A можна дістатися швидше! Оновімо вартість і батька.

$\text{costs}[n] = \text{new_cost}$
 ↑ ↑
 "FIN" 6

A	5
B	2
FIN	6

COSTS ←

$\text{parents}[n] = \text{node}$
 ↑ ↑
 "FIN" "A"

A	B
B	START
FIN	A

PARENTS ←

Функція `find_lowest_cost_node` вузол з найменшою вартістю знаходиться так:

```
def ind_lowest_cost_node(costs):
    lowest_cost = float("inf")
    lowest_cost_node = None
    for node in costs:
        cost = costs[node]
        if cost < lowest_cost and node not in processed:
            lowest_cost = cost
            lowest_cost_node = node
    return lowest_cost_node
```

Перебрать все узлы

...он назначается новым узлом с наименьшей стоимостью

Если это узел с наименьшей стоимостью из уже виденных и он еще не был обработан...

Реалізація на C++